Introduction to Hadoop and MapReduce

THE CONTRACTOR IS ACTING UNDER A FRAMEWORK CONTRACT CONCLUDED WITH THE COMMISSION

Eurostat

Large-scale Computation

- Traditional solutions for computing large quantities of data relied mainly on processor
  - **Complex processing made on data moved in memory**
  - **Scale only by adding power (more memory, faster processor)**
  - **Works for relatively small-medium amounts of data but cannot keep up with larger datasets**
- How to cope with today's indefinitely growing production of data?
  - **Terabytes per day**

Distributed Computing

- Multiple machines connected among each other and cooperating for a common job
- Challenges
  - **Complexity of coordination – all processes and data have to be maintained syncronized about the global system state**
  - **Failures**
  - **Data distribution**

Hadoop

- Open source platform for distributed processing of large data
  - **Based on a project developed at Google**
- Functions:
  - **Distribution of data and processing across machine**
  - **Management of the cluster**

- Simplified programming model
  - **Easy to write distributed algorithms**

Hadoop scalability

- Hadoop can reach massive scalability by exploiting a simple distribution architecture and coordination model
- Huge clusters can be made up using (cheap) commodity hardware
  - **A 1000-CPU machine would be much more expensive than 1000 single-CPU or 250 quad-core machines**
- Cluster can easily scale up with little or no modifications to the programs

Hadoop Concepts

- Applications are written in common high-level languages
- Inter-node communication is limited to the minimum
- Data is distributed in advance
  - **Bring the computation close to the data**
- Data is replicated for availability and reliability
- Scalability and fault-tolerance

Scalability and Fault-tolerance

- Scalability principle
  - **Capacity can be increased by adding nodes to the cluster**
  - **Increasing load does not cause failures but in the worst case only a graceful degradation of performance**
- Fault-tolerance
  - **Failure of nodes are considered inevitable and are coped with in the architecture of the platform**
  - **System continues to function when failure of a node occurs – tasks are re-scheduled**
  - **Data replication guarantees no data is lost**
  - **Dynamic reconfiguration of the cluster when nodes join and leave**

Benefits of Hadoop

- Previously impossible or impractical analysis
- Lower cost
- Less time
- Greater flexibility
- Near-linear scalability
- Ask Bigger Questions

Hadoop Components

Hive | Pig | Sqoop | HBase

Flume | Mahout | Oozie

Core Components

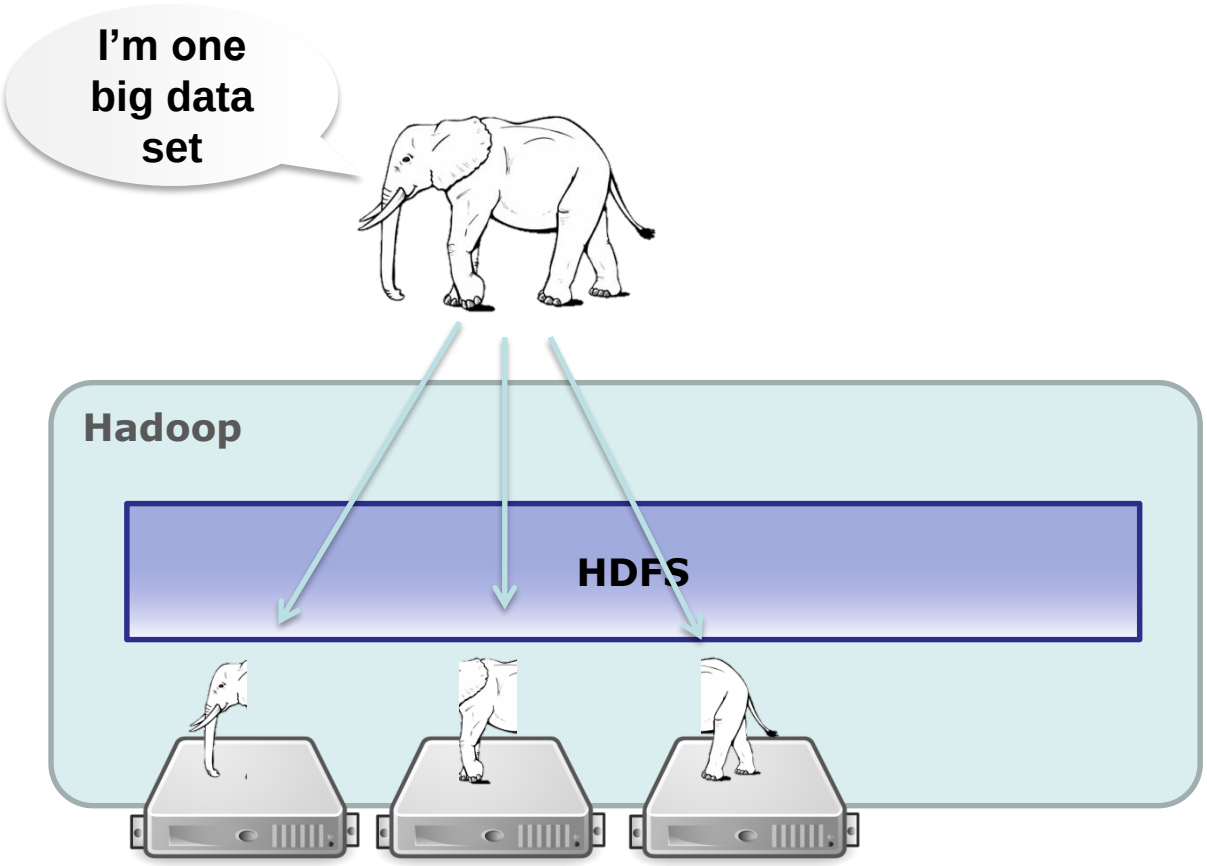
Hadoop Core Components

- HDFS: Hadoop Distributed File System
  - **Abstraction of a file system over a cluster**
  - **Stores large amount of data by transparently spreading it on different machines**
- MapReduce
  - **Simple programming model that enables parallel execution of data processing programs**
  - **Executes the work on the data near the data**
- In a nutshell: HDFS places the data on the cluster and MapReduce does the processing work

Structure of an Hadoop Cluster

- Hadoop Cluster:
  - **Group of machines working together to store and process data**
- Any number of "worker" nodes
  - **Run both HDFS and MapReduce components**
- Two "Master" nodes
  - **Name Node: manages HDFS**
  - **Job Tracker: manages MapReduce**

Hadoop Principle



Hadoop is basically a middleware platforms that manages a cluster of machines

The core components is a distributed file system (HDFS)

Files in HDFS are split into blocks that are scattered over the cluster

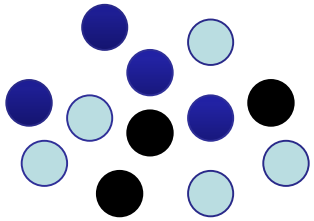
The cluster can grow indefinitely simply by adding new nodes

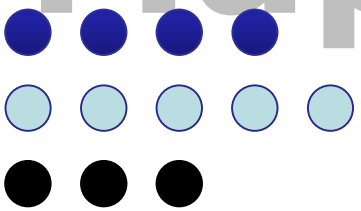
The MapReduce Paradigm

Parallel processing paradigm

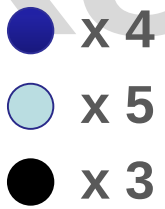
Programmer is unaware of parallelism

Programs are structured into a two-phase execution



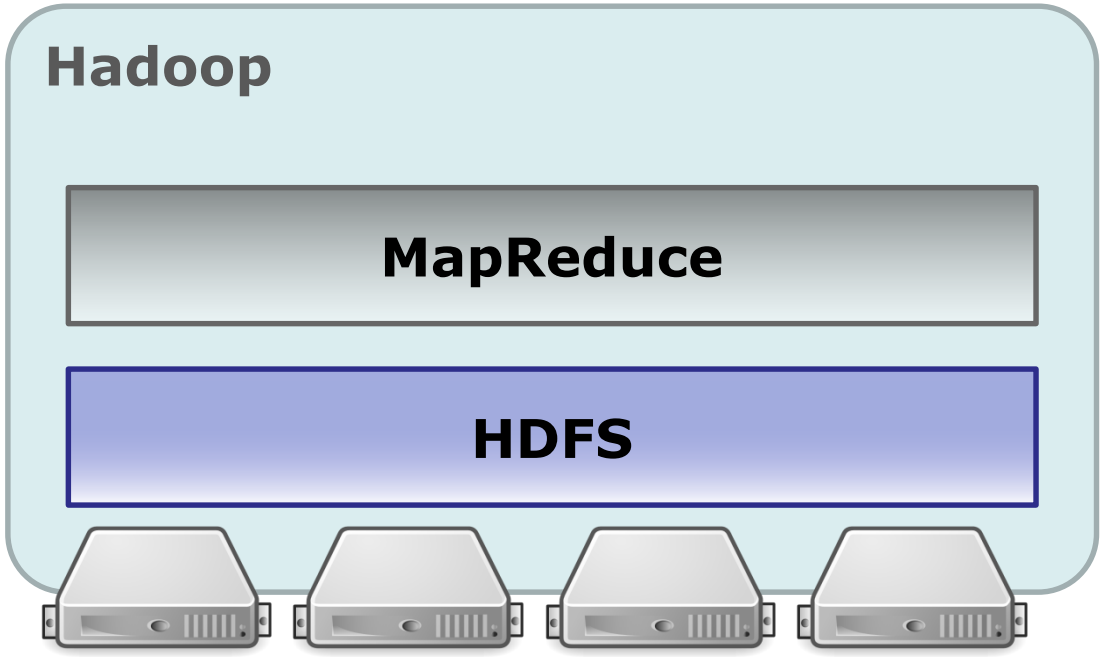


Data elements are classified into categories



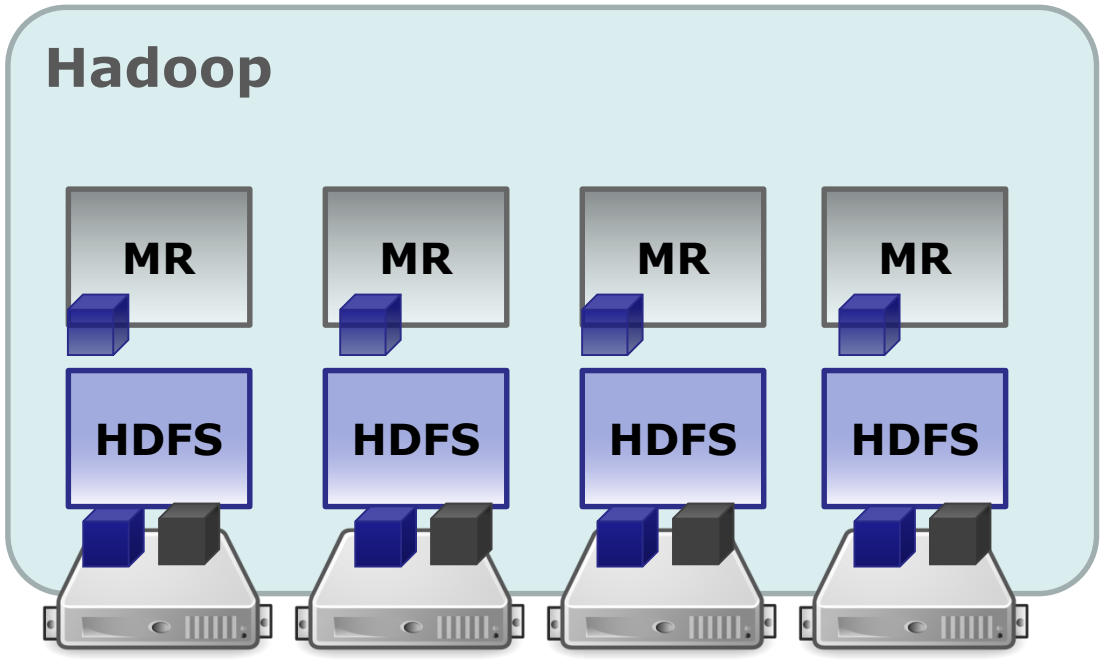
An algorithm is applied to all the elements of the same category

MapReduce and Hadoop



MapReduce is logically placed on top of HDFS

MapReduce and Hadoop



MR works on (big) files loaded on HDFS

Each node in the cluster executes the MR program in parallel, applying map and reduces phases on the blocks it stores

Output is written on HDFS

Scalability principle:
Perform the computation were the data is

HDFS Concepts

- Logical distributed file system that sits on top of the native file system of the operating system
  - **Written in Java**
  - **Usable by several languages/tools**
- All common commands for handling operations on a file systems are defined
  - **ls, chmod, …**
- Commands for moving files from/to the local file system are present

Accessing and Storing Files

- Data files are split into blocks and transparently distributed when loaded
- Each block is replicated in multiple nodes (default:3)
- NameNode stores metadata
- "Write-once" file write policy
  - **No random writes/updates on files are allowed**
- Optimized for streaming reads of large files
  - **Random reads are discouraged/impossible**
- Performs best with large files (>100Mb)

Accessing HDFS

- Command line

  ```
  >hadoop fs -<command>
  ```

- Java API
- Hue
  - **Web-based, interactive UI**
  - **Can browse, upload, download and view files**

MapReduce

- Programming model for parallel execution
- Programs are realized just by implementing two functions
  - **Map and Reduce**
- Execution is streamed to the Hadoop cluster and the functions are processed in parallel on the data nodes

MapReduce Concepts

- Automatic parallelization and distribution
- Fault-tolerance
- A clean abstraction for programmers
  - **MapReduce programs are usually written in Java**
  - **Can be written in any language using Hadoop Streaming**
  - **All of Hadoop is written in Java**
- MapReduce abstracts all the 'housekeeping' away from the developer
  - **Developer can simply concentrate on writing the Map and Reduce functions**

Programming Model

- A MapReduce program transforms an input list into an output list
- Processing is organized into two steps:

```
map  (in_key, in_value) ->
       (out_key, intermediate_value) list
```

```
reduce (out_key, intermediate_value list) ->
      out_value list
```

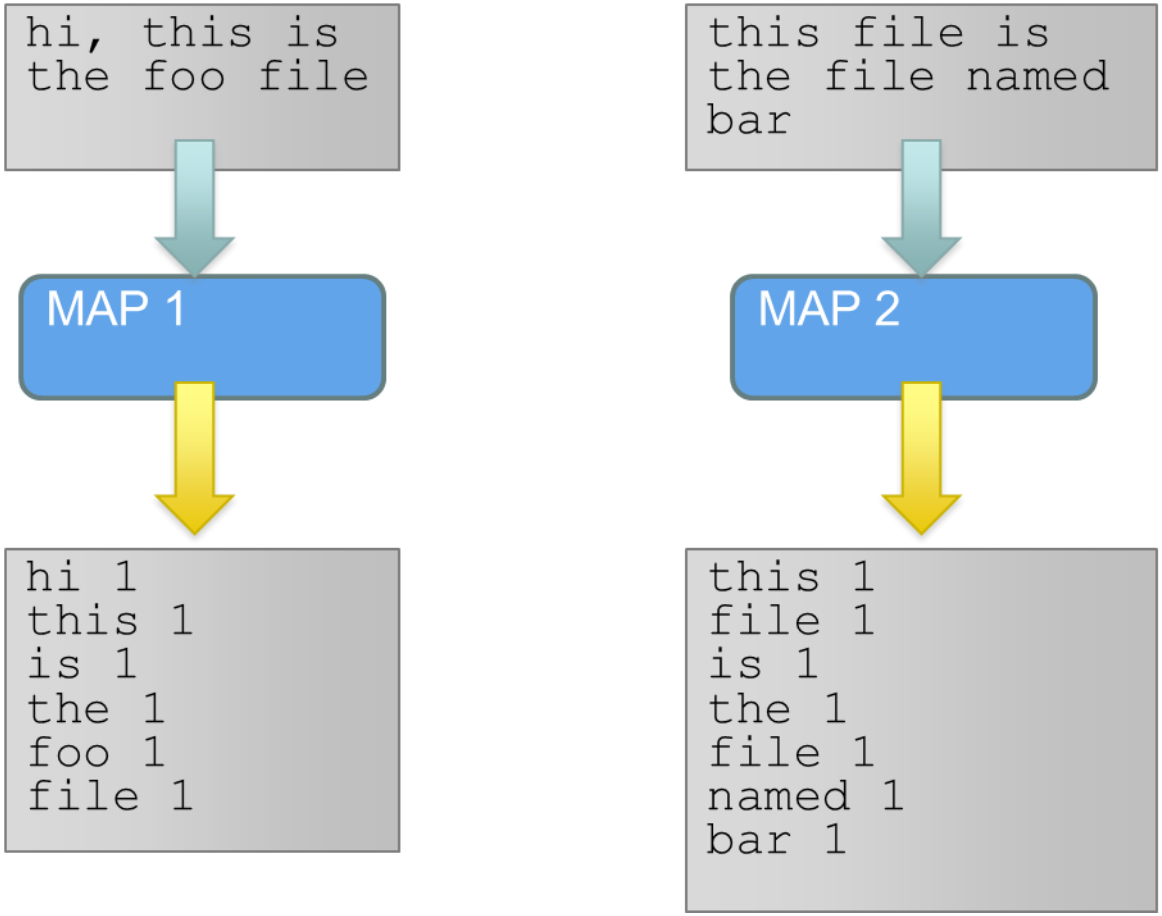
map

- Data source must be structured in records (lines out of files, rows of a database, etc)
- Each record has an associated key
- Records are fed into the map function as key*value pairs: e.g., (filename, line)
- map() produces one or more *intermediate* values along with an output key from the input
- In other words, map identifies input values with the same characteristics that are represented by the output key
  - **Not necessarily related to input key**

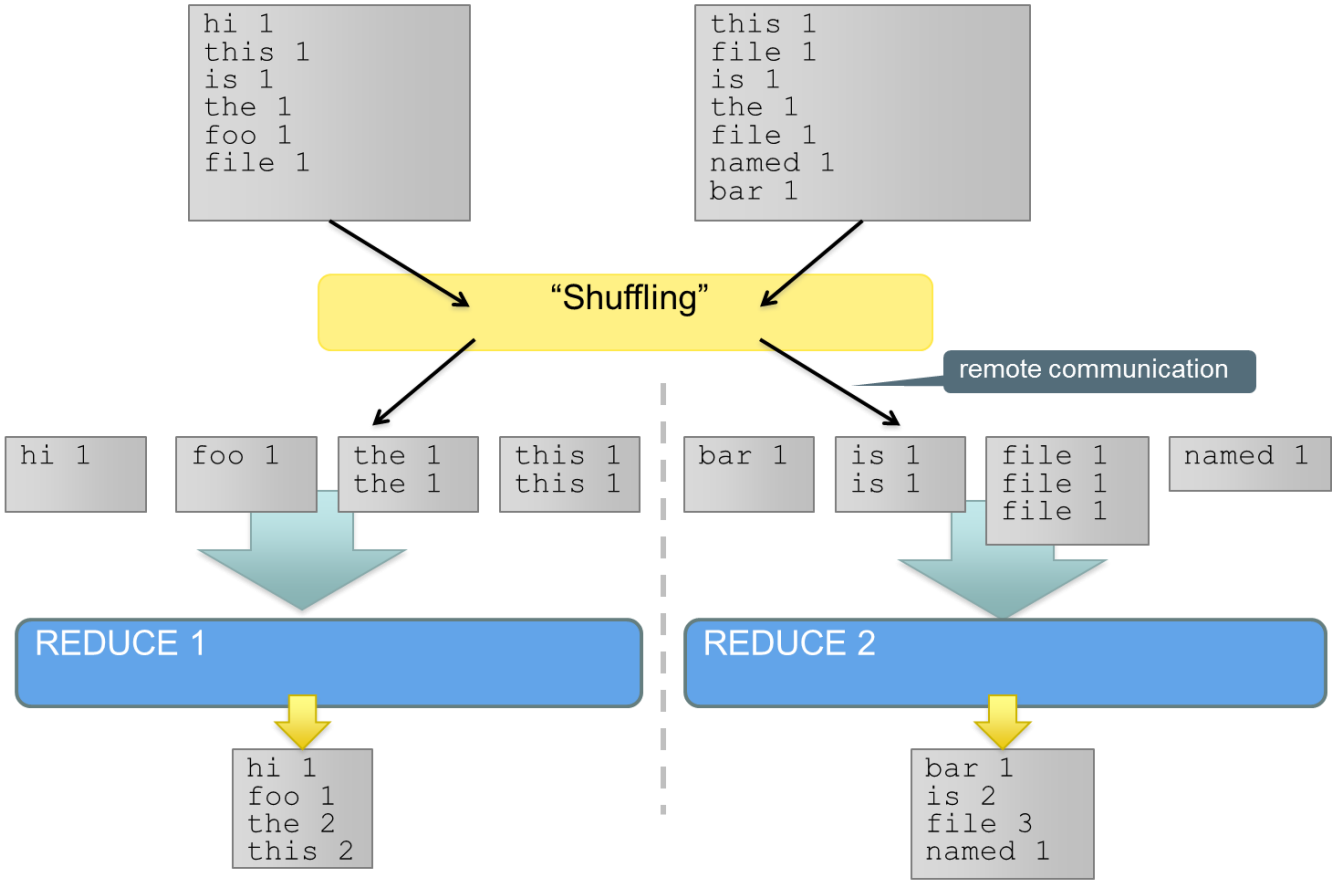
reduce

- After the map phase is over, all the intermediate values for a given output key are combined together into a list
- reduce() aggregates the intermediate values into one or more final values *for that same intermediate key*
  - **in practice, usually only one final value per key**

Example: word count



Example: word count



Beyond Word Count

- Word count is challenging over massive amounts of data
  - **Using a single compute node would be too time-consuming**
  - **Number of unique words can easily exceed available memory**
  - **Would need to store to disk**
- Statistics are simple aggregate functions
  - **Distributive in nature**
  - **e.g., max, min, sum, count**
- MapReduce breaks complex tasks down into smaller elements which can be executed in parallel
- Many common tasks are very similar to word count
  - **e.g., log file analysis**

MapReduce Applications

- Text mining
- Index building
- Graph creation and analysis
- Pattern recognition
- Collaborative filtering
- Prediction models
- Sentiment analysis
- Risk assessment